CSE166_FA23_assignment_2

October 11, 2023

1 CSE 166 Image Processing, Fall 2023 - Assignment 2

Instructor: Ben Ochoa

Assignment due: Wed, Oct 18, 11:59 PM

Name:

PID:

1.1 Instructions

1. All solutions to the problems below must be written in this notebook. Programming aspects of the assignment must be completed using Python (preferably 3.10). Please check the **README.md** for setup of the virtual environment.
2. This assignment must be completed **individually**. For more details, please review the Academic Integrity Policy and Collaboration Policy on Canvas.
3. The packages you need to complete the assignment will be indicated in the problem descriptions.You are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and TAs if you are unsure about the packages to use.
4. It is highly recommended that you begin working on this assignment early.
5. After finishing the assignment in the notebook, please export the notebook as a PDF and a python source file. You need to **submit 3 files: the Notebook, the PDF and the python file** (i.e. the `.ipynb`, the `.pdf` and the `.py` files) on Gradescope.

- To convert the notebook to PDF, you can choose one way below:
  - You may first export the notebook as HTML, and then print the web page as PDF
    - e.g., in Chrome: File → Save and Export Notebook as → "HTML"; or in VS-code: Open the Command Palette by pressing Ctrl+Shift+P (Windows/Linux) or Cmd+Shift+P (macOS), search for Jupyter: Export to HTML
    - Open the saved web page and right click → Print... → Choose "Destination: Save as PDF" and click "Save")
  - If you have XeTex installed on your machine, you may directly export the notebook as PDF: e.g., in Chrome, File → Save and Export Notebook as → "PDF"
  - You may use nbconvert to convert the ipynb file to pdf using the following command `jupyter nbconvert --allow-chromium-download --to webpdf filename.ipynb`

- To convert the notebook to python file, you can choose one way below:
  - You may directly export the notebook as py: e.g., in Chrome, File → Save and Export Notebook as → "Executable script"; or in VScode: Open the Command Palette and search for Jupyter: Export to Python Script
  - You may use nbconvert to convert the ipynb file to python file using the following command `jupyter nbconvert --to script filename.ipynb`

6.. Please make sure the content in each cell (e.g. code, output images, printed results, etc.) are clearly visible and are not cut-out or partially cropped in your final PDF file.

7. While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

2 Problem 1: Textbook problems (10 points)

Download the following textbook problems from the Resources tab on Piazza. **Answers to different versions of these problems will not be accepted.**

These textbook problems are conceptual and/or math problems, not programming problems. You are provided with a Markdown cell to answer each problem. Do not change this cell to a Code cell or create a new Code cell. **Answers in the form of code will not be accepted.**

2.1 a) Problem 3.5 (2 points)

Your answer here:

2.2 b) Problem 3.7 (2 points)

Your answer here:

2.3 c) Problem 3.11a (1 point)

Your answer here:

2.4 d) Problem 3.12a (1 point)

Your answer here:

2.5 e) Problem 3.24 (1 point)

Your answer here:

2.6 f) Problem 3.29 (2 points)

Note: Calculate the full padding convolution and use zero padding.

Your answer here:

2.7 g) Problem 3.44 (1 point)

Your answer here:

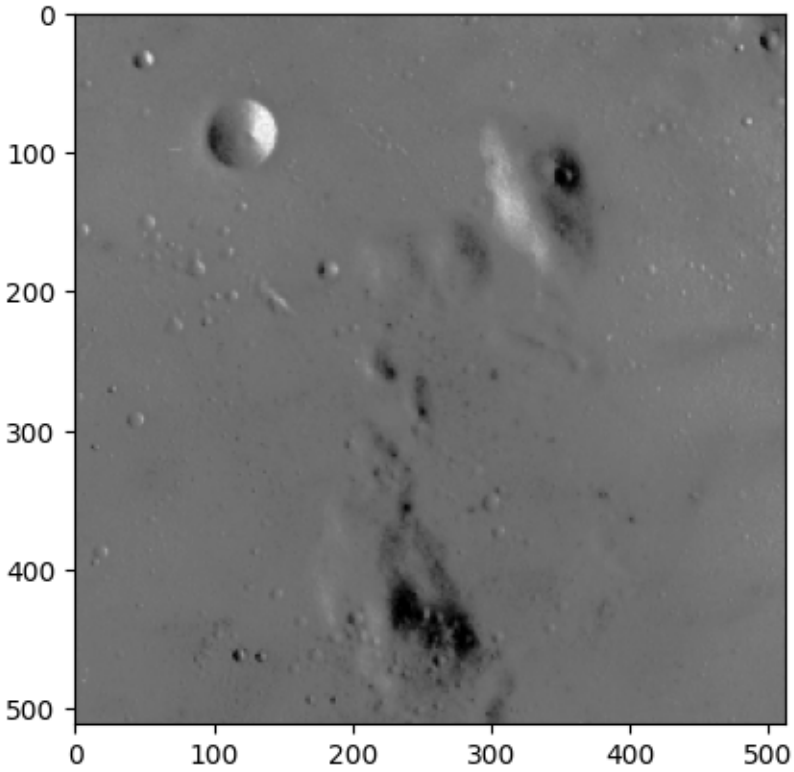
3 Problem 2: Programming: Intensity Transformations and Spatial Filtering (30 points)

3.1 Part 1: Piecewise Linear Transformation (5 points)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import math
```

[ ]:
```
# Read and display image
img = data.moon()

plt.imshow(img, cmap='gray')
plt.show()
```



1. Complete the function `piecewise_linear_transform` that computes an intensity transformed image of a given image. The function parameters are comprised of the input image to be transformed, a set of input intensity values, and a corresponding set of output intensity values. In both sets, the first intensity value is 0 and the last intensity value is 255. The resulting intensity values in the output image are calculated from a piecewise linear transformation determined from the parameters.

[ ]:
```
# function that transforms intensities of the image using piece-wise linear transform
def piecewise_linear_transform(img, ip_intensities, op_intensities):
  """
  img: input image (H,W)
  ip_intensities: input intensities (N,)
  op_intensities: output intensities (N,)

  returns:
  tr_img: resulting image after piecewise linear transform (H,W)
  """
  # your code here
  tr_img = []

  return tr_img
```

Call the function `piecewise_linear_transform` for the moon image with input intensities $\{0, 100, 150, 255\}$ and output intensities $\{0, 50, 200, 255\}$ to calculate the intensity transformed image. Display the input image and output image side by side.

[ ]:
```
"""
Call piecewise_linear_transform function for the moon image with the given␣
 ↪input
and output intensities.
Display the input and output images.
"""
ip_intensities = np.array([0, 100, 150, 255])
op_intensities = np.array([0, 50, 200, 255])
tr_img = piecewise_linear_transform(img, ip_intensities, op_intensities)

fig, ax = plt.subplots(1, 2, figsize=(10,10))
ax[0].imshow(img, cmap='gray')
ax[0].set_title('Input image')
ax[1].imshow(tr_img, cmap='gray')
ax[1].set_title('Transformed image')
plt.show()
```

2. Briefly discuss the qualitative differences between the input and output images.

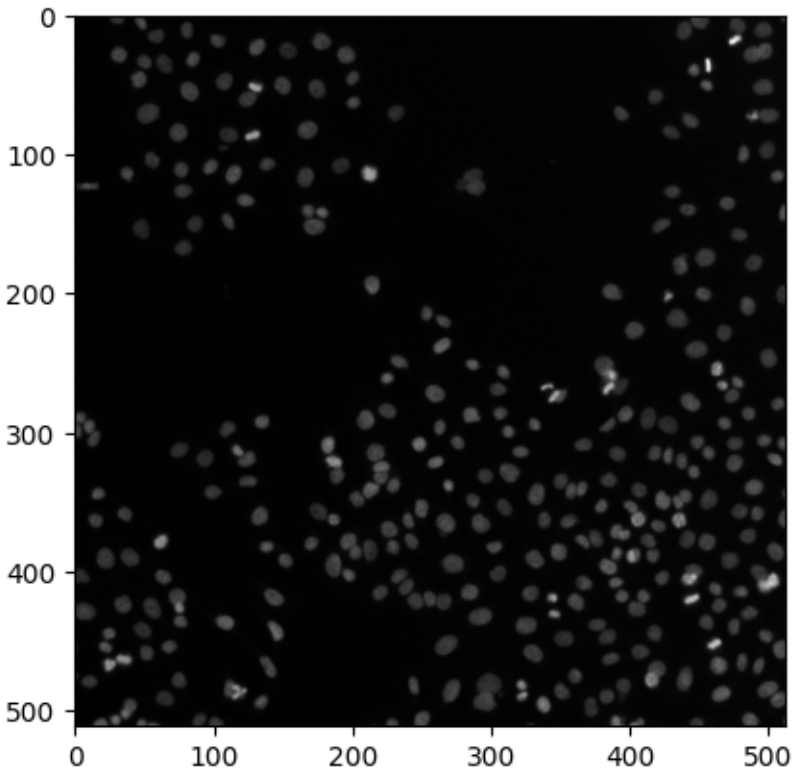
Your answer here:

3.2 Part 2: Histogram Equalization (5 points)

Read and display the human mitosis image.

[ ]:
```
# Read and display the image
img = data.human_mitosis()

plt.imshow(img, cmap='gray')
plt.show()
```



1. Complete the function `histogram_equalization` that calculates a histogram equalized image corresponding to a given input image. The function input is a grayscale image and the function output is the histogram equalized image corresponding to the input image.

[ ]:
```
# function that transforms the input image into histogram-equalized image
def histogram_equalization(img):
  """
  img: input image

  returns:
  heq_img: histogram-equalized image
  """
  # your code here
  heq_img = []

  return heq_img
```

Call the function `histogram_equalization` for the human-mitosis image. Display the input and output images. Clearly label the images and set the value range to [0, 255] on all display calls.

[ ]:
```
"""
Call histogram_equalization for the human-mitosis image.
Display the input and output images.
```

```
"""
# your code here
```

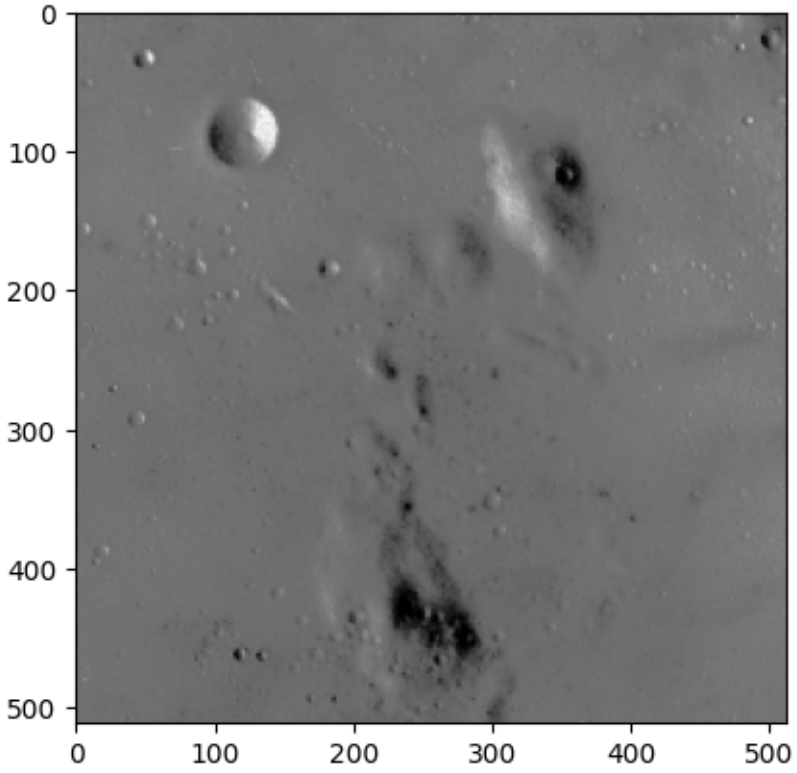
2. Plot the unnormalized histograms of the input image and the histogram-equalized image side-by-side. Set the number of equal-width bins of the histograms as 16.

[ ]:
```
# your code here
```

3.3 Part 3: Sharpening using the second derivative (10 points)

[ ]:
```
# Read and display the image
img = data.moon()

plt.imshow(img, cmap='gray')
plt.show()
```



1. Complete the function `sharpen_image` that computes a sharpened image corresponding to a given input image. The function input is a grayscale image and the function output is the sharpened image corresponding to the input image. The function must use the Laplacian filter (without diagonal directions). Note that $g(x, y) = f(x, y) - \nabla^2 f(x, y)$, where $f(x, y)$ is the input image and $g(x, y)$ is the output sharpened image. Filtering must be implemented **without** using Python in-built library functions like *scipy.ndimage.convolve*.

Important Notes:
* Implement the 'same' padding convolution ,i.e. the output image should have the same size as the input image. Apply replicate padding, i.e. pad with the edge values of the image. * While performing sharpening, the output pixels can have values outside the range [0,255]. You may handle this by working with `np.int32` data type matrices. * Clamp the output sharpened image such that all intensities less than zero is set to 0 and all intensities greater than 255 is set to 255.

[ ]:
```
# function that sharpens the given image
def sharpen_image(img):
  """
  img: input image (H,W)

  returns:
  sh_img: image after sharpening (H,W)
  """
  # your code here
  sh_img = []

  return sh_img
```

2. Call the function `sharpen_image` for the microaneurysms image. Display the input image and output image side by side. Clearly label the images and set the value range to [0, 255] on all display calls.

[ ]:
```
"""
Call the sharpen_image function with the microaneurysms image.
Display the input and output images.
"""
# your code here
```

3. Briefly discuss your results.

Your answer here:

4. **Optional**: Sharpen the image using the Laplacian filter with diagonal directions and briefly discuss the qualitative differences between these results and those obtained using the filter without diagonal directions.

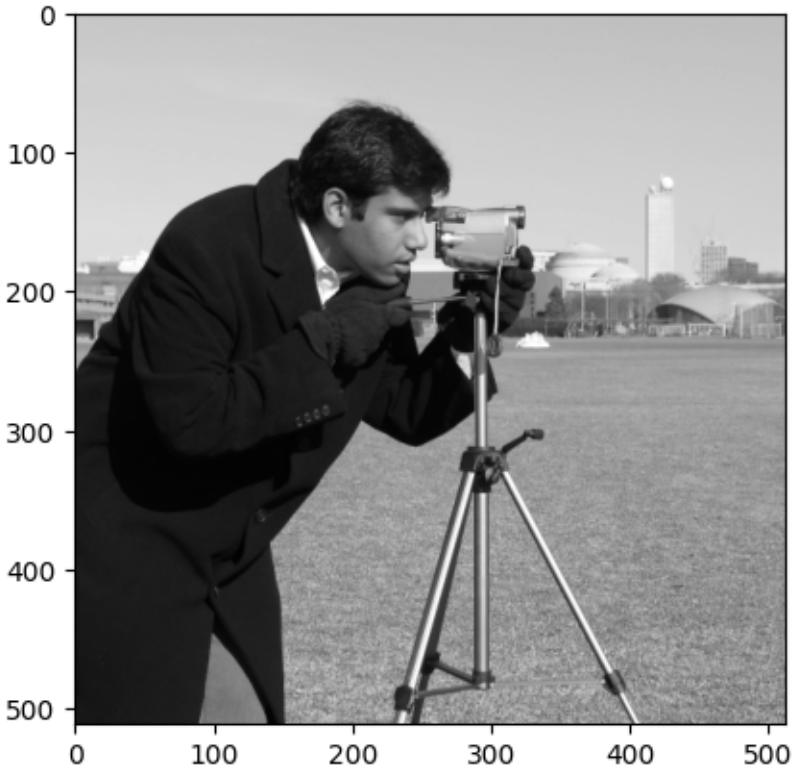
[ ]:
```
# your code here (optional)
```

Your answer here (optional):

3.4 Part 4: Magnitude of gradient (10 points)

[ ]:
```
# Read and display the image
img = data.camera()

plt.imshow(img, cmap='gray')
plt.show()
```



1. Complete the function `gradient_magnitude` that calculates the magnitude of gradient vector image corresponding to a given input image. The function input is a grayscale image and the function output is the gradient magnitude image corresponding to the input image. Use the **Sobel operator** to compute the derivative. Filtering must be implemented without using Python in-built functions like *scipy.ndimage.convolve*.

Important Note: * Implement the 'same' padding convolution ,i.e. the output image should have the same size as the input image. Apply replicate padding, i.e. pad with the edge values of the image. * While computing magnitude of gradient, the output pixels can have values outside the [0,255] range and/or floating-point numbers. You can handle this by working with `np.float32` data type matrices. * Scale the intensity of the output image such that the maximum value in the gradient magnitude image is 255.

[ ]:
```
# function that computes the magnitude of gradient for the given image using␣
↪the sobel operator
def gradient_magnitude(img):
  """
  img: input image (H,W)

  returns:
  mg_img: image after applying magnitude of gradient (H,W)
  """
```

```
    # your code here
    mg_img = []

    return mg_img
```

Call the function `gradient_magnitude` with the camera image. Display the input image and output image side by side. Clearly label the images and set the value range to [0, 255] on all display calls.

[ ]:
```
"""
Call the gradient_magnitude function with the camera image.
Display the input and output images.
"""
# your code here
```